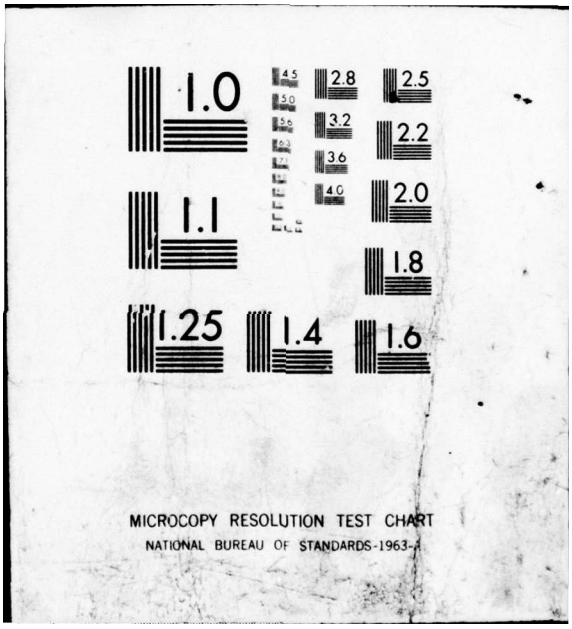
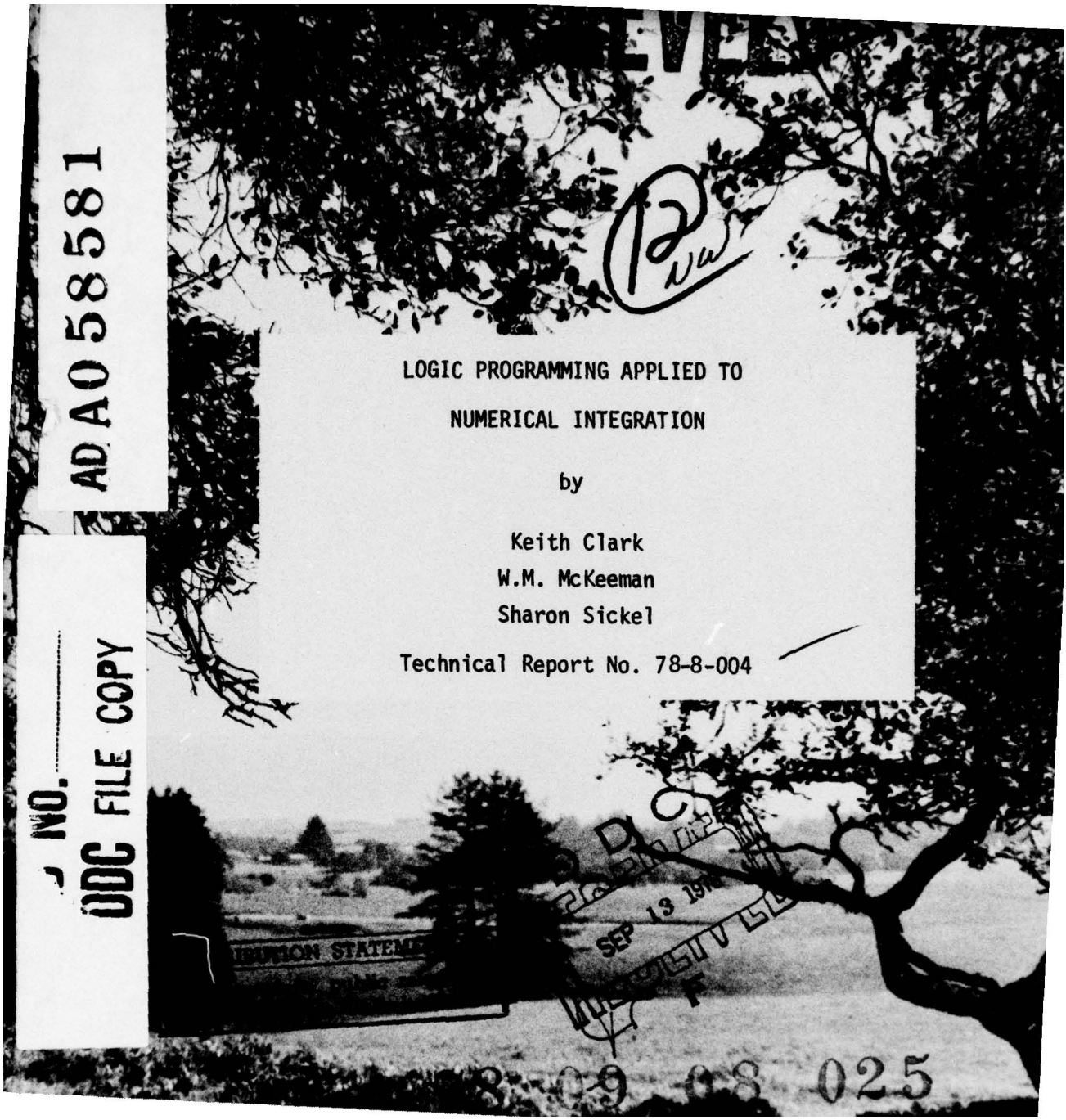
AD-A058 581 CALIFORNIA UNIV SANTA CRUZ INFORMATION SCIENCES F/G 9/2
LOGIC PROGRAMMING APPLIED TO NUMERICAL INTEGRATION.(U)
AUG 78 K CLARK, W M MCKEEMAN, S SICKEL N00014-76-C-0682
UNCLASSIFIED TR-78-8-004 NL

1 OF 1
AD A058581

END
DATE FILMED
11-78
DDC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A



LOGIC PROGRAMMING APPLIED TO NUMERICAL INTEGRATION

by

Keith Clark
W.M. McKeeman
Sharon Sickel

Technical Report No. 78-8-004

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER

2. GOVT ACCESSION NO.

3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle)

(6) LOGIC PROGRAMMING APPLIED TO NUMERICAL INTEGRATION

5. TYPE OF REPORT & PERIOD COVERED

(9) Technical rept.

6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s)

(10) Keith Clark
W.M. McKeeman
Sharon Sickel

8. CONTRACT OR GRANT NUMBER(s)

(15) N00014-76-C-0682

9. PERFORMING ORGANIZATION NAME AND ADDRESS

Information Sciences
University of California
Santa Cruz, California 95064

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS

(11) 1 Aug 78

11. CONTROLLING OFFICE NAME AND ADDRESS

Office of Naval Research
Arlington, Virginia 22217

12. REPORT DATE

August 1, 1978

13. NUMBER OF PAGES

22

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)

Office of Naval Research
University of California
553 Evans Hall
Berkeley, California 94720

15. SECURITY CLASS. (of this report)

Unclassified

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

(12) 27 p.

This document has been approved for public release and sale; its distribution is unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Distribution of this document is unlimited. It may be released to the Clearinghouse, Department of Commerce, for sale to the general public.

18. SUPPLEMENTARY NOTES

(14) TR-78-8-004

19. KEY WORDS (Continue on reverse side if necessary and identify by block number) and Phrases:

Programming methods, specification of algorithms, logic programming, numerical integration, Romberg integration, adaptive integration.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This paper uses logic programming to describe numerical integration in a general way. Axioms are added to the basic logic programs to define specific, known numerical integration algorithms. Using a slightly different formalization, we construct logic programs for adaptive Romberg integration and for a new algorithm for adaptive integration.

The logic programs provide a formal basis for classifying numerical quadrature algorithms (recall Rice's comment that there are 1,000,000 useful ones).

(continued on next page)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-LF-014-6601

78 09 08 025

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

420 350

20. (Continued)

The logic programs are also more understandable than the corresponding programs in conventional programming languages.

In terms of logic programming there are several novel points in this paper. The data type, real number, is not defined by a recursive constructor function. Termination is decided by error bounds, rather than by reaching some basis case of the data constructor. The problem itself seems to demand concurrent execution and global variables but in fact is solved in a linear fashion.

LOGIC PROGRAMMING APPLIED TO NUMERICAL INTEGRATION

Keith Clark

Department of Computer Science and Statistics

Queen Mary College

University of London

W. M. McKeeman

Sharon Sickel

Information Sciences

and

Crown College

University of California

Santa Cruz

This work was supported by Office of Naval Research Contract # 76-C-0682

ABSTRACT

This paper uses logic programming to describe numerical integration in a general way. Axioms are added to the basic logic programs to define specific, known numerical integration algorithms. Using a slightly different formalization, we construct logic programs for adaptive Romberg integration and for a new algorithm for adaptive integration.

The logic programs provide a formal basis for classifying numerical quadrature algorithms (recall Rice's comment that there are 1,000,000 useful ones). The logic programs are also more understandable than the corresponding programs in conventional programming languages.

In terms of logic programming there are several novel points in this paper. The data type, real number, is not defined by a recursive constructor function. Termination is decided by error bounds, rather than by reaching some basis case of the data constructor. The problem itself seems to demand concurrent execution and global variables but in fact is solved in a linear fashion.

1.1 Review of Adaptive Quadrature

The numerical approximation of integrals goes back to the early Greeks and the approximation of π. An excellent survey of the problem and modern solution methods is found in Numerical Integration [2].

Ultimately, all numerical approximations take the form

$$\int_b^a f(x)dx = (b-a)\sum_{k=0}^{n-1} w_k f(x_k)$$

where n is the number of intervals, and where

$$\sum_{k=0}^{n-1} w_k = 1.$$

That is, the approximation is a weighted sum of function values over the interval of integration. The problems are:

1. choosing the sample points (x_k)
2. choosing the weights (w_k)
3. knowing how accurate the result is.

Adaptive algorithms cluster the points x_k within regions where the intergrand is most intractable. The strategies for doing so, and the resulting programs, have tended to become increasingly opaque and complex [4].

It is usual to present only deterministic numerical algorithms. They vary by the kind of approximation used, the order of applying approximations, and the strategy for termination. If we broaden our view to include nondeterministic algorithms, we can arrange them hierarchically; the lower in the hierarchy, the more deterministic the algorithm. The hierarchy defines families of algorithms and is a natural result of using logic programming for specification.

1.2 Review of Logic Programming

Logic programs are a subset of well-formed-formulas (WFs) of first-order predicate calculus that define a function or relation and that can be used to drive the computation of a function or one or more missing values of a relation [3,8].

The WFs of logic programs are restricted to the clausal form

$$B \leftarrow A_1 \wedge A_2 \ldots A_n$$

where the A_i's are predicates of the form

$$\text{name}(\text{parameter}_1, \text{parameter}_2, \ldots, \text{parameter}_m)$$

and B is either a predicate or empty. Each parameter is either a constant, a variable, or a simple function of variables or constants. The variables are all assumed to be universally quantified. It has been shown [1] that all WFs of first-order predicate

calculus are formalizable within this restriction.

The procedural interpretation of logic programs is as follows:

$$\leftarrow A_1 \wedge \ldots \wedge A_n$$

is the main program which is successfully executed when each A_i is known to be true. Each A_i is interpreted as a procedure call.

$$B \leftarrow A_1 \wedge A_2 \ldots A_n$$

is a procedure definition. B is the head including the procedure name and its parameters; the A_i's are the procedure body and correspond to procedure calls as in the main program.

A call is accomplished by matching the parameters of the call, A_i, to a procedure head B of the same name, binding the parameters of the call and the procedure head. Each call has the effect of establishing the A_i's as subgoals.

$$B \leftarrow$$

is a basis case where, having been called, it establishes no further subgoals. In the predicate sense, B ← is an assertion that B is true for the parameters given.

A call $P(x_1,\ldots,x_k)$ is carried out by finding a procedure definition of the form

$$P(y_1,\ldots,y_k) \leftarrow Q_1 \wedge Q_2 \wedge \ldots \wedge Q_n$$

and then unifying (x_i,y_i) $1 \leq i \leq k$. This has the effect of assigning values to the local variables of the procedure, but may also cause values to be passed to the calling procedure.

Example. Consider the logic program for the factorial function:§

1) FACT(0,1) ←
2) FACT(n+1,z) ← FACT(n,x) ∧ TIMES(n+1,x,z)
3) ← FACT(2,ANS)

---

§ FACT(x,y) is true if and only if factorial(x) = y.
TIMES(x,y,z) is true if and only if x·y = z.

Clauses 1) and 2) are two separate procedures that together define the factorial function. Clause 1) is the procedure that terminates the computation by assigning the output to be 1 if the input is 0. Clause 2) is the procedure that reduces a factorial computation to a simpler factorial and a multiplication. We assume here that TIMES has been appropriately defined by other axioms or is considered a primitive function. The form of the parameters will determine which procedure definition is actually used for each call. Clause 3) is the initial call and serves the role of forcing a computation of factorial(2) and leaving the answer in variable ANS.

Let's follow the computation. FACT(2,ANS) is a call to a procedure. It cannot call clause 1) because 2 and 0 cannot be made to unify since they are two distinct constants.[§] So a call to clause 2) is made. FACT(2,ANS) must be made to match FACT(n+1,z). This can happen if we bind n to 1 and ANS to z. The procedure with local variables assigned looks like:

$$FACT(2,ANS) \leftarrow FACT(1,x) \wedge TIMES(2,x,ANS)$$

i.e. ANS = 2·factorial(1). This gives us two new calls:

$$FACT(1,x) \quad \text{and} \quad TIMES(2,x,ANS).$$

[§] Normally data types, e.g. the natural numbers used here, are defined constructively, as Peano did, e.g.

$$NATNUM(0)$$
$$NATNUM(n) \rightarrow NATNUM(s(n)).$$

Then clause 2) would be:

$$FACT(s(n),z) \leftarrow FACT(n,x) \wedge TIMES(s(n),y,z).$$

The above can either be interpreted as a shortcut notation for Peano's axioms, or for efficiency, we can take the natural numbers and a few functions on natural numbers as primitives.

Let us first consider only the calls to FACT. FACT(1,x) cannot call clause 1) because constants 0 and 1 will not unify; clause 2) is called again. The matching process produces:

$$FACT(1,x) \leftarrow FACT(0,x') \wedge TIMES(1,x',x)$$

which, in turn, produces two new calls. Considering again only the call on FACT: FACT(0,x') calls clause 1). Clause 2 will be excluded if we have properly defined the natural numbers, i.e. $0 \neq n+1$. After the binding, the procedure produced from clause 1 is:

$$FACT(0,1) \leftarrow$$

which is unchanged, but the binding binds 1 with x' in the calling procedure. Since clause 1) has no body, we simply return. Now TIMES(1,1,x) is able to be called and binds 1 with x, giving a fully defined procedure

$$FACT(1,1) \leftarrow FACT(0,1) \wedge TIMES(1,1,1).$$

In addition, x is now bound in the previous stage and as we return we find

$$FACT(2,ANS) \leftarrow FACT(1,1) \wedge TIMES(2,1,ANS).$$

TIMES again is called to yield ANS = 2, and finally FACT(2,2) as desired.

We use arithmetic expressions in the parameter lists for ease of readability. It is possible to compute such expressions using predicates, e.g.

$$(\frac{m-a}{b-a})\cdot\varepsilon$$

could be computed:

$$SUB(m,a,x) \wedge SUB(b,a,y) \wedge DIV(x,y,z) \wedge TIMES(z,\varepsilon,\varepsilon_1)$$

where

SUB(u,v,w) is true iff u-v = w

DIV(u,v,w) is true iff u/v = w

TIMES(u,v,w) is true iff $z\cdot\varepsilon = \varepsilon_1$.

The shorthand notation is only syntactic sugar and does not change the essential

character of logic programs.

The implementation of logic programming systems raises some points not directly relevant to this paper, i.e., how, in general, to select procedures, what order to evaluate subgoals, how to achieve efficiency, and so on. To some extent our logic programs reflect these considerations, where we use the logic itself to force selection and order. The reader interested in this aspect should read Warren's description of an operational logic program compiler interpreter [9].

2.1 Formal Specification of Adaptive Quadrature

The specification is a set of axioms which predetermine the desired result. The first is a formal identity from the calculus.

$$\int_a^b f(x)dx = \int_a^m f(x)dx + \int_m^b f(x)dx$$

From the above we can derive the following theorem:

Thm.

$$\left|\int_a^m f(x)dx - y_1\right| \le \varepsilon_1 \wedge \left|\int_m^b f(x)dx - y_2\right| \le \varepsilon_2 \qquad (1)$$

$$\Rightarrow \left|\int_a^b f(x)dx - (y_1+y_2)\right| \le \varepsilon_1+\varepsilon_2$$

Proof: $\quad |s| \le x \wedge |t| \le y \Rightarrow |s+t| \le x+y.$

Formula (1) gives the basic step of the adaptive routine, reducing the whole problem to two, presumably simpler, subgoals. The subdivision process terminates when it is determined that a numerical approximation can be used instead. There are many ways to make the determination, each depending upon a pair of formulas, named here quad and err.

For example, given the basic trapezoidal approximation

$$trap(f,a,b) = (f(a)+f(b))\cdot(b-a)/2$$

we may use

$$quad(f,a,b) = trap(f,a,\tfrac{a+b}{2}) + trap(f,\tfrac{a+b}{2},b)$$

and

$$err(f,a,b) = \left|quad(f,a,b) - trap(f,a,b)\right|$$

We make the assumption:

$$\left|\int_a^b f(x)dx - quad(f,a,b)\right| \leq err(f,a,b) \qquad (2)$$

While neither formula (1) nor formula (2) is true in general, they are two of the assumptions upon which numerical quadrature routines are based and therefore have the status of axioms here.

So, given interval (A,B), function F, and error bound EPS, we would like to find a y such that

$$\left|\int_A^B F(x)dx - y\right| \leq EPS \qquad (3)$$

We accomplish this by subdividing the interval and error bound, as described in (1') below unless err(f,a,b) is less than or equal to the error bound allotted for that interval, in which case we evaluate using quad and then return that answer to aid in building up the integral for the next inclusive interval.

The structure of the problem can be seen more easily perhaps if it is written in predicate form. Define the predicate:

$$I(a,b,f,\varepsilon,y) \Leftrightarrow \left|\int_a^b f(x)dx - y\right| \leq \varepsilon$$

Then consider the following:

Logic Program 1:

$$I(a,b,f,\varepsilon_1+\varepsilon_2,y_1+y_2) \leftarrow err(f,a,b) > \varepsilon_1+\varepsilon_2 \quad (1')$$
$$\wedge\ I(a,m,f,\varepsilon_1,y_1)$$
$$\wedge\ I(m,b,f,\varepsilon_2,y_2)$$

$$I(a,b,f,\varepsilon,quad(f,a,b)) \leftarrow err(f,a,b) \leq \varepsilon \quad (2')$$

$$\leftarrow I(A,B,F,EPS,answer) \quad (3')$$

(1') is the recursive case; (2') is the termination case; (3') is the main program, i.e. it is a call to I for inputs A, B, F, EPS, and output answer.

The above logic program describes, mathematically, a large class of integration algorithms. However, the methods for splitting the interval and the error bound are unspecified and the computation will be highly nondeterministic. Specifying the splitting methods will give us particular algorithms, and, in general, more determinism. In addition, we may not even get closer to a solution at each step. Suppose, for example, that we always chose a "midpoint", m, outside the interval (a,b). Then instead of reducing the interval size at each step, we may actually be increasing it. Nothing in the logic program, as it stands, contradicts this behavior. So, let us consider some ways to increase the efficiency of this algorithm by further specification.

Given a clause C and a property p, we can tighten or restrict C by p by replacing clause C by clause C $\leftarrow$ p. For example, to require that "midpoint" m be strictly between points a and b, change clause (1') to:

$$(1') \leftarrow (a < m < b)$$

which is equivalent to adding the restriction to the list of subgoals of (1'):

$$I(a,b,f,\varepsilon_1+\varepsilon_2,y_1+y_2) \leftarrow err(f,a,b) > \varepsilon_1+\varepsilon_2$$
$$\wedge\ I(a,m,f,\varepsilon_1,y_1)$$
$$\wedge\ I(m,b,f,\varepsilon_2,y_2)$$
$$\wedge\ (a < m < b)$$

Consider the particular algorithm in which the intervals and error bounds are both halved.

Logic Program 2:

$$I(a,b,f,\varepsilon,y_1+y_2) \leftarrow err(f,a,b) > \varepsilon$$
$$\wedge\ I(a,\tfrac{a+b}{2},f,\tfrac{\varepsilon}{2},y_1)$$
$$\wedge\ I(\tfrac{a+b}{2},b,f,\tfrac{\varepsilon}{2},y_2)$$

plus (2') and (3').

If we want to modularize the program, so that there is a procedure to decide how to split the interval and error bound, we could write the program:

Logic Program 3:

$$I(a,b,f,\varepsilon,y) \leftarrow err(f,a,b) > \varepsilon$$
$$\wedge\ I(a,m,f,\varepsilon_1,y_1)$$
$$\wedge\ I(m,b,f,\varepsilon_2,y_2)$$
$$\wedge\ MID(a,b,m)$$
$$\wedge\ ER\_DIV(a,b,m,\varepsilon,\varepsilon_1,\varepsilon_2)$$

together with (2') and (3'), and procedures defining MID and ER_DIV.

If we are dividing the intervals and the error bounds proportionately, as in the earliest adaptive quadrature routines [5], then we could define ER_DIV:

$$ER\_DIV(a,b,m,\varepsilon,\varepsilon_1,\varepsilon_2) \leftarrow \varepsilon_1 = (\tfrac{m-a}{b-a})\cdot\varepsilon$$
$$\wedge\ \varepsilon_2 = (\tfrac{b-m}{b-a})\cdot\varepsilon$$

If we define

$$MID(a,b,m) \leftarrow m = \frac{a+b}{2}$$

then with this definition of ER_DIV, Logic Program 2 and 3 are equivalent, as can easily be proven. Note that the equality symbols represent equality and not assignment. Procedure ER_DIV expresses a relation that holds among its arguments; the value computed depends upon which arguments are already bound. E.g. if you called ER_DIV with values for a, b, ε, ε_1, then it could compute m and ε_2. For more discussion of varying the orientation of functions, see [3,7].

3. Global Algorithms

The usual measure of efficiency is the number of times the function is evaluated in the quadrature summation. Assumptions 1, 2 & 3 define a class of adaptive quadrature algoritnms. While it would be nice to choose an optimal member from the class, it is not possible. Rice [6] claims that there are at least 1,000,000 algorithms, each optimal in its own domain. Furthermore, by the Mean Value Theorem, one function evaluation is enough for continuous integrands, if only one knew how to choose the single sample point in the interval.

There are other measures of efficiency, in particular the amount of memory needed, which can be applied. Measures of this sort are implementation dependent.

Nevertheless, one can pick a strategy which is both intuitively attractive and performs well experimentally. Since at each splitting of a sub-interval the estimate of the truncation error over the whole sub-interval is known, we may choose to split the pending sub-interval having largest error estimate. This strategy is new as far as known to the authors and is presented below.

The next two algorithms are considered global because there is a sense of having to know about many subgoals at once.

In this algorithm, we continue to use the notion of subdividing intervals, but

instead of distributing the error bound evenly over the interval, we would like to reserve most of the error for the trouble spots by taking advantage of the accuracy in the easy spots. We think of the interval having been broken into sub-intervals already, and associated with each sub-interval is an approximation of the integrand and an error approximation. If the sum of the error approximations is greater than the total error bound allowed (ε) across the interval, then we divide the sub-interval having the largest error estimate. We redefine I to use this method, and we call I as before. Logic Program 4 is a formal specification for this algorithm.

Logic Program 4:

```
I(a,b,f,ε,area) ←
    G(f,ε,{(a,b,err(f,a,b))},err(f,a,b),area)
G(f,ε,intset,errsum,area) ← errsum ≤ ε
                ∧ ΣQUAD(intset,f,area)
G(f,ε,intset,errsum,area) ← errsum > ε
                ∧ GREATEST(intset,(a,b,e))
                ∧ MID(a,b,m)
                ∧ intset' = intset
                          - {(a,b,e)}
                          U {(a,m,err(f,a,m)),(m,b,err(f,m,b))}
                ∧ errsum' = errsum-e + err(f,a,m) + err(f,m,b)
                ∧ G(f,ε,intset',errsum',area)
```

The semantics of the predicates used in Logic Program 4 are given in Table 1. Logic program definitions of ΣQUAD and GREATEST are given below.

Compute the sum of quad applied to all elements of an INTSET.

$$\Sigma QUAD(\phi,f,0) \leftarrow$$

$$\Sigma QUAD(T,f,area) \leftarrow T = T' \cup \{(a,b,e)\}$$
$$\wedge\ T - \{(a,b,e)\} = T' \quad \S$$
$$\wedge\ \Sigma QUAD(T',f,area')$$
$$\wedge\ area = (area' + quad(f,a,b))$$

Find a sub-interval with the greatest error approximation.

$$GREATEST(\{(a,b,e)\},(a,b,e)) \leftarrow$$

$$GREATEST(T,(a,b,e)) \leftarrow$$
$$T = T' \cup \{(a_1,b_1,e_1)\}$$
$$\wedge\ T - \{(a_1,b_1,e_1)\} = T'$$
$$\wedge\ GREATEST(T',(a_2,b_2,e_2))$$
$$\wedge\ MAX((a_1,b_1,e_1),(a_2,b_2,e_2),(a,b,e))$$

where

$$MAX((a_1,b_1,e_1),(a_2,b_2,e_2),(a_1,b_1,e_1)) \leftarrow e_1 \geq e_2$$
$$MAX((a_1,b_1,e_1),(a_2,b_2,e_2),(a_2,b_2,e_2)) \leftarrow e_1 < e_2$$

The predicate G will compute exactly the same values, in the same order, as one of the (nondeterministic) paths of Logic Programs 1 or 3. If $MID(a,b,m) \leftarrow m = \frac{a+b}{2}$, then G will compute exactly the same values as one of the paths of Logic Program 2.

---

§ These first two conditions are needed to create a choice function for sets. They mean slightly different things and both are required.

predicate	semantics
G(f,ε,intset,errsum,area)	Value, area, is the sum of the integrals over the intervals of intset under function f, within error ε. Value, errsum, is the sum of the error components of the elements of intset. Errsum is redundant, since we could recompute it each time, but it is useful both for efficiency and clarity.
ΣQUAD(intset,f,area)	$\left[\sum_{(a,b,e) \in \text{intset}} \text{quad}(f,a,b) \right] = \text{area}$
GREATEST(intset,(a,b,e))	(a,b,e) is an element of intset having the highest third component, i.e. error approximation.
MID(a,b,m)	as used before, given interval (a,b) this procedure selects a midpoint m for subdivision.

Table 1

Program 4 can be improved by reducing the number of evaluations of f and quad. Every time err(f,a,b) is called, quad(f,a,b) must be evaluated, which forces evaluation of f(a),f((a+b)/2) and f(b). The quad and f values are computed redundantly and are discarded. The next program saves f and quad values in such a way that for any points a and b, f(a) and quad(a,b) are computed at most once. The saved values make the resulting formulas a little more cumbersome, but no more complex.

Four other changes have been made to Program 4:

1) Intervals are expressed as left endpoint and width rather than left and right endpoints.
2) Instead of keeping a set of intervals and finding the one with the greatest error estimate, we keep a list of intervals in decreasing order of error estimate. The symbol ⊗ denotes an infix "cons" of an element to a list.
3) In order to prevent redundant computations of f in deriving the value of quad, we have to fix the quadrature rule; we have chosen the trapezoidal rule, but we could have chosen others.
4) ΣQUAD is redefined to use the information that has been saved in intlist.

Logic Program 5:

```
I(a,b,f,ε,area) ←
    G(f,ε,(a,h,fa,fm,fb,q,e),e,area)
            ∧ h = b-a
            ∧ fa = f(a)
            ∧ fm = f((a+b)/2)
            ∧ fb = f(b)
            ∧ q = (fa + 2fm + fb)·h/4
```

(Logic Program 5 Cont'd):

$G(f,\varepsilon,intlist,errsum,area) \leftarrow errsum \leq \varepsilon$

$\quad \wedge\ \Sigma QUAD(intlist,area)$

$G(f,\varepsilon,intlist,errsum,area) \leftarrow errsum > \varepsilon$

$\quad \wedge\ intlist = (a,h,fa,fm,fb,q,e) \otimes list$

$\quad\quad \wedge\ fm_1 = f(a + h/4)$

$\quad\quad \wedge\ q_1 = (fa + 2fm_1 + fm)\cdot h/8$

$\quad\quad \wedge\ e_1 = |q_1 - ((fa + fm)\cdot h/4)|$

$\quad \wedge\ MERGE(list,(a,h/2,fa,fm_1,fm,q_1,e_1),list')$

$\quad\quad \wedge\ fm_2 = f(a + 3h/4)$

$\quad\quad \wedge\ q_2 = (fm + 2fm_2 + fb)\cdot h/8$

$\quad\quad \wedge\ e_2 = |q_2 - ((fm + fb)\cdot h/4)|$

$\quad \wedge\ MERGE(list',(a + h/2,h/2,fm,fm_2,fb,q_2,e_2),list'')$

$\quad\quad \wedge\ errsum' = errsum - e + e_1 + e_2$

$\quad \wedge\ G(f,\varepsilon,list'',errsum',area)$

$\Sigma QUAD((),0) \leftarrow$

$\Sigma QUAD((a,h,fa,fm,fb,q,e) \otimes list\ ,\ q + area)$

$\quad \leftarrow \Sigma QUAD(list,area)$

$MERGE((),i,i) \leftarrow$

$MERGE((a_1,h_1,fa_1,fm_1,fb_1,q_1,e_1) \otimes list\ ,\ (a_2,h_2,fa_2,fm_2,fb_2,q_2,e_2),$
$\quad (a_2,h_2,fa_2,fm_2,fb_2,q_2,e_2) \otimes (a_1,h_1,fa_1,fm_1,fb_1,q_1,e_1) \otimes list)$
$\quad \leftarrow e_2 \geq e_1$

$MERGE((a_1,h_1,fa_1,fm_1,fb_1,q_1,e_1) \otimes list\ ,\ (a_2,h_2,fa_2,fm_2,fb_2,q_2,e_2),$
$\quad (a_1,h_1,fa_1,fm_1,fb_1,q_1,e_1) \otimes list')$
$\quad \leftarrow e_2 < e_1$
$\quad \wedge\ MERGE(list,(a_2,h_2,fa_2,fm_2,fb_2,q_2,e_2)\ ,\ list')$

4. Romberg Integration

One can improve the accuracy of the integral over a given set of sample points by either resorting to higher order rules or by applying the Romberg convergence formulas to the trapezoidal rule values. Romberg is preferable because it is numerically stable [2]. Romberg cannot be applied over arbitrarily spaced sample points; in its usual form the interval of integration must be uniformly divided into 2^k sub-intervals for some exponent k. This property has generally precluded its use in adaptive routines.

In the algorithms previously described, there may be sub-intervals which are in fact subdivided exactly 2^k times for some k. Romberg can be applied to them giving improved approximations. Then all of the approximations can be summed to give the final result, which is almost always an improvement over the unaccelerated values. There is no new information on the error estimate so it remains the same. The adaptive Romberg algorithm is the same as Logic Program 5 in the subdivision of the intervals, but different in the computation of the integral. It is intuitively attractive to identify the powers of 2 arising naturally out of the subdivision process with those needed by Romberg, particularly where recursion is used since the calling routine contains some values of use in the Romberg rule. The algorithm presented here does not make that identification and as a result, is better able to apply the Romberg rule. The reason is that 2^k contiguous sub-intervals may break across the recursive structure but are available in the lists used here.

Logic Program 6:

Change the termination case (clause 2) of Logic Program 5 to:

$$G(f,\varepsilon,intlist,errsum,area) \leftarrow errsum \leq \varepsilon$$
$$\wedge\ SORT(intlist,intlist')$$
$$\wedge\ EVAL(0,(),0,intlist',area)$$

and add the following procedures:

1) Procedure SORT(x,y) takes list of intervals x and sorts it based on the first value of the 7-tuple representing each interval to form list y, i.e. the left endpoint of the interval. The actual program for sort is not included here. It is a simple program, and operates similarly to MERGE.

2) Procedure EVAL(left,mid,intsize,valist,area) takes a sorted list (valist) of intervals and determines if there are any sequences of sub-intervals (mid) to which Romberg acceleration can be applied. Any adjacent intervals of equal width (intsize) are eligible.[§] EVAL collects in "mid" contiguous sequences of equal sized intervals in modified form, then sends them to REDUCE for evaluation. The value, left, is the sum of the Romberg approximations of the integrals of the intervals to the left of mid. The value, area, is the sum of left, and the Romberg approximations of the integrals of the intervals in mid and valist.

```
EVAL(left,mid,intsize,(),left + x)
     ← REDUCE((),mid,x)
EVAL(left,mid,intsize,(a,h,fa,fm,fb,qamb,e) ⊗ valist,area)
     ← intsize = h
     ∧ qab = (fa + fb)·h/2
     ∧ qlist = qab ⊗ qamb
     ∧ EVAL(left,mid ⊗ (h,fa,fb,qlist),intsize,valist,area)
EVAL(left,mid,intsize,(a,h,fa,fm,fb,qamb,e) ⊗ valist,area)
     ∧ intsize ≠ h
     ∧ qab = (fa + fb)·h/2
     ∧ qlist = qab ⊗ qamb
     ∧ EVAL(left + x,(h,fa,fb,qlist),h,valist,area)
     ∧ REDUCE((),mid,x)
```

---

§ The equality test on real numbers makes sense here because if the numbers being compared are mathematically equal, they are computed exactly the same way (repeated division by 2) hence will also be numerically equal.

3) Procedure REDUCE($list_1$,$list_2$,area) takes the trapezoidal approximations provided by EVAL and computes the values needed to apply Romberg acceleration, that is to say the zero-level Romberg values, by pairwise combining contiguous intervals of the same step size until a single interval is constructed. The formula

$$T^k(a,h) + T^k(a+h,h) = T^{k+1}(a,h)$$

where $T^k(a,h)$ is the trapezoidal approximation on interval (a,a+h) with 2^k+1 sample points provides the basic computation of REDUCE. The final result is a list of the form

$$T^0(a,h) \otimes T^1(a,h) \otimes T^2(a,h) \ldots \otimes T^n(a,h) .$$

REDUCE pairwise combines the intervals of $list_2$ and places the result in $list_1$. When $list_2$ is exhausted, $list_1$ and $list_2$ are exchanged and the process repeated until the list has been reduced to a single interval. If the starting list is not of length equal to a power of two, there will be left-over intervals when the intervals are pairwise combined. These are passed to ROMB at that time, thus are not considered for further Romberg acceleration.

REDUCE((),(),0) ←

REDUCE(ilist,(h,fa,fb,qlist),$area_1$+$area_2$)
← ROMB(1,qlist,$area_1$)
∧ REDUCE((),ilist,$area_2$)

REDUCE(ilist,(),area) ← ilist ≠ ()
∧ REDUCE((),ilist,area)

REDUCE(ilist,(h,fa,fm,$qlist_1$) ⊗ (h,fm,fb,$qlist_2$) ⊗ ilist',area)
← qab = (fa + fb)·h/2
∧ SUMLIST($qlist_1$,$qlist_2$,$qlist_3$)
∧ qlist = qab ⊗ $qlist_3$
∧ REDUCE(ilist ⊗ (2·h,fa,fb,qlist),ilist',area)

where SUMLIST($qlist_1$,$qlist_2$,$qlist_3$) takes two lists, $qlist_1$ and $qlist_2$, and pairwise sums their corresponding components to form $qlist_3$.

SUMLIST((),(),()) ←

SUMLIST(q ⊗ L,r ⊗ L',(q+r) ⊗ x)

← SUMLIST(L,L',x)

4) Procedure ROMB(k,qlist,area) takes the list provided by REDUCE, which is also the set of values

$$R_0^0(a,h) \otimes R_1^0(a,h) \otimes R_2^0(a,h) \ldots \otimes R_n^0(a,h)$$

where $R_n^k(a,h)$ is the $k^{\underline{th}}$ acceleration on interval (a,a+h) using 2^n+1 function values. We have

$$R_n^k = (4^k R_n^{k-1} - R_{n-1}^{k-1})/(4^k-1) \text{ for } k>0$$

and $R_n^n(a,h)$ is the desired result. The parameters of ROMB are k, the level of acceleration, qlist, the list of values at that state, and area, the ultimate answer.

ROMB(k,(),r,r) ← length(r) = 1

ROMB(k,L,r_1 ⊗ r_2 ⊗ R,area)

← y = $(4^k r_2 - r_1)/(4^k-1)$

∧ ROMB(k,L ⊗ y,r_2 ⊗ R,area)

ROMB(k,L,r,area) ← L≠() ∧ length(r) = 1

∧ ROMB(k+1,(),L,area)

5. Correctness

The logic programs given are verifiable since each procedure is an easily proved theorem, (in fact many are theorems found in numerical analysis texts). For example, Logic Program 1 consists of two procedures that correspond to the two theorems:

Thm.

$$(\mathrm{err}(f,a,b) \leq \varepsilon) \rightarrow \left| \int_a^b f(x)dx - \mathrm{quad}(f,a,b) \right| \leq \varepsilon$$

pf.

formula (2) and transitivity

Thm.

$$\left.\begin{array}{l} (\mathrm{err}(f,a,b) > \varepsilon) \\ \wedge \left| \int_a^m f(x)dx - y_1 \right| \leq \varepsilon_1 \\ \wedge \left| \int_m^b f(x)dx - y_2 \right| \leq \varepsilon_2 \end{array}\right\} \rightarrow \left| \int_a^b f(x)dx - (y_1 + y_2) \right| \leq \varepsilon_1 + \varepsilon_2$$

pf.

Theorem of section 2.

So long as the compiler/interpreter preserves the logical meaning, we are assured of correctly integrating.

To have total correctness, we must also prove that the algorithms terminate. If the looping or recursion of a program is governed by counting down a natural number

to zero, or decomposing a constructed object until an atomic or primitive object is reached, then you can easily prove termination. In the examples here, the reduction of an error term to a specified bound causes the algorithms to stop. The proofs of termination, therefore, are more difficult here. Proof of termination of Logic Programs 2 and 3 can be achieved by proving:

> Given interval (A,B) and function F, there exists an interval size d such that for all a and b such that $(A \leq a \leq b \leq B) \wedge (b-a) \leq d$
>
> $$err(F,a,b) \leq \varepsilon \cdot (\frac{b-a}{B-A}).$$

5. Conclusions

We have shown several numerical integration algorithms and their specifications in logic. The logic programs are unusually concise, compared with most specifications of numerical integration algorithms. Each procedure must represent a theorem in numerical analysis. Partial correctness is thus assured, by definition. Termination is trickier, both in deriving a mathematical guarantee of termination and termination when dealing with truncated reals, as on a computer.

We have extended logic programs to use on real numbers and error-bound termination, and except for proof of termination and deciding how to reduce a problem into subproblems, this use of logic programming is not significantly different from logic algorithms on constructively defined data types.

REFERENCES

1. Andreka, H. and Nemeti, J. "The Generalized Completeness of Horn Predicate-Logic as a Programming Language". D.A.I. Research Report No. 21, Department of Artificial Intelligence, University of Edinburgh (March 1976).

2. Davis, Philip J. and Rabinowitz, Philip. Numerical Integration. Blaisdell Publishing Co., Waltham, Mass. (1967).

3. Kowalski, Robert. Predicate Logic as Programming Language. Information Processing 74, North-Holland Publishing (1974).

4. Lyness, J.N. SQUANK (Simpson quadrature used adaptively, noise killed). Algorithm 379, Comm. ACM 13,4 (April 1970) 260-263.

5. McKeeman, W.M. Adaptive Numerical Integration by Simpson's Rule. Algorithm 145, Comm. ACM 5,12 (December 1962) 604.

6. Rice, John R. A Metalgorithm for Adaptive Quadrature, J.ACM 22,1 (January 1975) 61-82.

7. Sickel, Sharon. Invertibility of Logic Programs. Technical Report 78-8-005, Information Sciences, University of California, Santa Cruz, Ca. (1978)

8. van Emden, M.H. and Kowalski, R.A. The Semantics of Predicate Logic as a Programming Language. J.ACM 23,4 (October 1976) 733-742.

9. Warren, David H.D. Implementing PROLOG -- compiling predicate logic programs. D.A.I. Research Reports 39 & 40, Department of Artificial Intelligence, University of Edinburgh (May 1977).

OFFICIAL DISTRIBUTION LIST

Contract N00014-76-C-0682

Defense Documentation Center
Cameron Station
Alexandria, VA 22314
12 Copies

Office of Naval Research
Information Systems Program
Code 437
Arlington, VA 22217
2 Copies

Office of Naval Research
Code 200
Arlington, VA 22217
1 Copy

Office of Naval Research
Code 458
Arlington, VA 22217
1 Copy

Office of Naval Research
Branch Office, Boston
Bldg. 114, Section D
666 Summer Street
Boston, MA 02210
1 Copy

Office of Naval Research
Branch Office, Chicago
536 South Clark Street
Chicago, ILL 60605
1 Copy

Office of Naval Research
Branch Office, Pasadena
1030 East Green Street
Pasadena, CA 91106
1 Copy

Naval Research Laboratory
Technical Information Division
Code 2627
Washington, D.C. 20375
6 Copies

Dr. A. L. Slafkosky
Scientific Advisor
Commandant of the Marine Corps (Code RD-1)
Washington, D.C. 20380
1 Copy

Naval Ocean Systems Center
Advanced Software Technology Division
Code 5200
San Diego, CA 92152
1 Copy

Mr. E. H. Gleissner
Naval Ship Research & Development Center
Computation and Mathematics Department
Bethesda, MD 20084
1 Copy

Captain Grace M. Hopper (008)
Naval Data Automation Command
Washington Navy Yard
Building 166
Washington, D.C. 20374
1 Copy